Construction and randomization of regular factorial designs with the R package planor

Hervé Monod and Annie Bouvier

MaIAGE, INRA, Université Paris-Saclay, 78350 Jouy-en-Josas, France

November 16, 2016

Contents

1 Introduction

The design of factorial experiments is an essential part of applied statistics. It is presented in many textbooks but for the planor user, we particularly recommend the book [2] by R.A. Bailey and the chapter [7] by A. Kobilinsky (in French). In practice, the experimenter aims to study the joint influence of several factors of interest, called the *treatment factors*, on one or more response variables. Before the experiment, the experimenter and — ideally — the statistician define the treatment factors and their levels. Based on the available resources and on the experimental protocol, they determine the size of the experiment and they may identify additional factors called nuisance or *block factors*, to be taken into account even though they are not interesting *per se*. Last but not least, they must anticipate how the results will be analysed or, more precisely, which model will be applied to the data. Based on all these specifications, they can search for a valuable class of designs, and then generate and randomize one such design for the experiment.

The R package planor is made to generate factorial designs by following this approach. To generate a design with planor, the user must provide information on the design factors, on the design size, and on the anova model to be used when analysing the results. The user then asks planor to search for one or more solutions to these specifications. The solutions returned by planor, if any, are orthogonal designs which ensure that all the factorial effects of interest will be estimable when the specified anova model will be applied to the experimental data. The designs can be randomized according to the relevant blocking structure, so that the estimators will be unbiased even if the model is only approximately true. Besides, orthogonality properties guarantee that the variance of the estimators will be minimal compared to other designs of the same size.

Strictly speaking, planor generates *regular* factorial designs, which are obtained by an algebraic method of construction and which form a sub-class of orthogonal designs. The algorithm implemented in planor thus diverges from an optimal design approach and it excludes a lot of interesting non-orthogonal designs. However, it includes many of the most useful classes of designs, such as complete block designs, Latin squares, split-plot designs, and many other full and fractional factorial designs. It can find designs for a small or a large number of factors, with possibly different numbers of levels. It can take into account hierarchical relationships among factors and control the confounding of treatments effects with block effects, like in split-plot or criss-cross experiments.

The main algorithm implemented in planor is presented in [12], together with the theory behind and with several detailed examples. This algorithm is based on the key matrix method introduced by [15] and refined by [1],[6]. It produces the so-called *regular* designs in which factorial effects are either estimable independently or completely confounded. The method is further described in [7], [10], [11] and illustrated in [4]. The initial planor manual [8] has been adapted to the planorR package [9] and it also gives more information on the theory than this guide.

This vignette first presents how planor can be used to generate easily ready-to-use factorial designs with different blocking structures. Section 2 also shows how the designs can be randomized. Then Section 5 gives more technical information on the objects generated by the planor functions. Indeed, the solutions, if any, can be obtained as a list of design key matrices. Several specific functions then allow to investigate the solutions' properties and to print and store the resulting designs.

Please note that planor is still under construction. More details are available through the help functions. We advise to check that the designs obtained by planor behave as expected before using them for a real experiment, by inspecting them and conducting analysis on simulated data, for example.

To start working with planor, we first load the package as usual.

```r
library("planor")

## Loaded planor 1.3.0
```

2 Two simple examples to illustrate the basic approach

2.1 Construction of a randomized complete block design

To illustrate the construction of a factorial design with `planor`, we first consider the classical case of a variety trial in agriculture. We assume that the experimenter wants to compare five varieties denoted by $V1$ to $V5$. No more than 20 plots are available and, because of the field heterogeneity, they are divided into four blocks of size five. The best solution, in that case, is to construct a randomized complete block design.

2.1.1 Step by step construction

The design construction may be decomposed into four generic main steps.

First step: specify the factors and their levels, using the function `planor.factors`. A first possibility is to give the names of the $s = 2$ factors to the `factors` argument, and their s numbers of levels to the `nlevels` argument:

```
trial.fac <- planor.factors(factors = c("Block", "Variety"),
    nlevels = c(4, 5))
```

In that case, the levels of the factors are the integers from 1 to their numbers of levels. Alternatively, alphanumeric levels can be specified by giving a list of vectors to the `factors` argument. Each element of the list must bear the name of a factor and contain the levels of that factor:

```
trial.fac <- planor.factors(factors = list(Block = 1:4,
    Variety = c("V1", "V2", "V3", "V4", "V5")))
```

Second step: specify the expected properties of the design, using the function `planor.model`. The simplest way to use the function `planor.model` is to give an anova formula to its `model` argument, involving the factors declared in the first step. `planor` will then search for designs which ensure that all the factorial effects in this formula can be fully estimated when it is applied to the response data with, typically, the R function `aov`.

For a complete block design, the usual model for the analysis contains the main effects of the block and treatment factors, whereas their interaction is neglected and confounded with the residual. So the model to be specified to `planor.model` is:

```
trial.mod <- planor.model(model = ~Block + Variety)
```

Third step: specify the design size and run the search for a solution, using the function `regular.design`. This function requires the information on the factors now stored in `trial.fac`, the information on the expected model now stored in `trial.mod` and the information on the design size given by the compulsory `nunits` argument. In the present step-by-step approach, `regular.design` is called as follows:

```
trial.plan <- regular.design(factors = trial.fac, model = trial.mod,
    nunits = 20, output = "data.frame")

## The search is closed: max.sol = 1 solution(s) found

print(t(trial.plan), quote = FALSE)

##         1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
## Block   1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4  4  4
## Variety V1 V2 V3 V4 V5 V1 V2 V3 V4 V5 V1 V2 V3 V4 V5 V1 V2 V3 V4 V5
```

The `output` argument is optional. By setting `output="data.frame"`, we ask `regular.design` to return the plan as a simple dataframe. We print just below the design in a more concise matrix form where each row corresponds to a block.

```
print(t(matrix(trial.plan$Variety, 5, 4)), quote = FALSE)

##      [,1] [,2] [,3] [,4] [,5]
## [1,] V1   V2   V3   V4   V5
## [2,] V1   V2   V3   V4   V5
## [3,] V1   V2   V3   V4   V5
## [4,] V1   V2   V3   V4   V5
```

Fourth step: randomize the design, using the function `planor.randomize`. In planor, the randomization is defined as a random permutation of the units with respect to an initial design and a specified block structure. The arguments of `planor.randomize` are `blockformula` to specify the block structure and `data` to give the initial design.

The randomization of a block design consists of a random permutation of the blocks and independent random permutations of the units within each block. For the variety trial, the syntax is:

```
trial.rand <- planor.randomize(blockformula = ~Block/UNITS,
    data = trial.plan)
print(t(matrix(trial.rand$Variety, 5, 4)), quote = FALSE)

##      [,1] [,2] [,3] [,4] [,5]
## [1,] V5   V1   V4   V3   V2
## [2,] V5   V2   V3   V4   V1
## [3,] V1   V3   V4   V2   V5
## [4,] V2   V4   V5   V3   V1
```

This is the final step of the design generation. Each row i of `trial.rand` gives the combination of levels of the factors which must be used in run (or plot) i of the experiment.

2.1.2 One-step construction

The step-by-step construction just described can be shortcut by giving all the required information directly to the function `regular.design`. To get the same design as above (except for the randomization step), the equivalent one-step syntax is:

```
trial.rand2 <- regular.design(factors = list(Block = 1:4,
    Variety = c("V1", "V2", "V3", "V4", "V5")), model = ~Block +
    Variety, nunits = 20, randomize = ~Block/UNITS, output = "data.frame")

## The search is closed: max.sol = 1 solution(s) found

print(t(matrix(trial.rand2$Variety, 5, 4)), quote = FALSE)

##      [,1] [,2] [,3] [,4] [,5]
## [1,] V4   V3   V2   V5   V1
## [2,] V2   V1   V5   V4   V3
## [3,] V1   V2   V3   V5   V4
## [4,] V1   V4   V3   V2   V5
```

2.1.3 Analysis

After the experiment is completed, the response variables can be added as new columns of the dataframe `trial.rand`. The data can be analysed by anova, using the specified model formula and the `aov` function. Based on simulated data, the analysis of the complete block trial gives:

```
trial.rand$Y <- runif(20)
trial.aov <- aov(Y ~ Block + Variety, data = trial.rand)
summary(trial.aov)

##             Df Sum Sq Mean Sq F value Pr(>F)
## Block        3 0.1937 0.06457   0.963  0.442
## Variety      4 0.6390 0.15975   2.382  0.110
## Residuals   12 0.8047 0.06706
```

2.2 Construction of a randomized Latin square design with two treatment factors

The complete randomized block design is a full factorial design, which means that all combinations of levels of the factors are replicated the same number of times. To show a less trivial example of what planor can do, we complete this introductory section by a design construction which involves more factors and cannot allow for a full factorial design. On the contrary, it requires to search for a fractional design, that is, a design which is too small to include all the combinations of levels of the factors.

Consider a sensory experiment to compare the six cake preparations which result from three recipes cooked at two different temperatures. The experimenter is interested in the main effects of both factors and in their interaction. We assume that the experiment consists of six tasters and six tasting periods. All tasters can assess all preparations consecutively as in a complete block design but, in addition, the experimenter wants to account for a possible period effect when analysing the data.

2.2.1 One-step construction

This problem can be solved by using a Latin square design. A Latin square is a table of n rows and n columns. It contains n symbols, with each symbol represented exactly once per row and once per column. With a Latin square, it is possible to study a treatment factor of interest while allowing for two block factors that are completely crossed. From a statistical point of view, it is a fractional factorial design for three factors at n levels in n^2 units, with all three factors being pairwise orthogonal. Here, the block factors are the tasters and the periods, and the n treatments are the combinations of the recipe and temperature factors.

In planor, this design can be generated and randomized automatically by the step-by-step or one-step constructions presented in Subsection 2.1. The one-step R code is:

```
TasterLevels <- paste("Tastor_", 1:6, sep = "")
PeriodLevels <- paste("Period", 1:6, sep = "")
LS.rand <- regular.design(factors = list(tastor = TasterLevels,
    period = PeriodLevels, recipe = LETTERS[1:3], temp = 1:2),
    model = ~tastor + period + recipe * temp, nunits = 36,
    randomize = ~tastor + period, output = "data.frame")

## The search is closed: max.sol = 1 solution(s) found
## The search is closed: max.sol = 1 solution(s) found
```

When declaring the factors, we show different possibilities based on basic R to specify their sets of levels. Note that the randomization of a Latin square consists in randomly permuting both the row and the coluns formula takes into account both the tastor and period block factors.

We print the first five rows of the design dataframe and then the design in its Latin square form :

```
print(LS.rand[1:5, ])

##      tastor  period recipe temp
## 1  Tastor_1 Period1      A    2
## 2  Tastor_1 Period2      B    1
## 3  Tastor_1 Period3      C    2
## 10 Tastor_1 Period4      A    1
## 11 Tastor_1 Period5      B    2

LS.rand$preparation <- paste(LS.rand$recipe, LS.rand$temp,
    sep = ".")
LS.table <- matrix(LS.rand$preparation, 6, 6)
rownames(LS.table) <- PeriodLevels
colnames(LS.table) <- TasterLevels
print(LS.table, quote = FALSE)

##         Tastor_1 Tastor_2 Tastor_3 Tastor_4 Tastor_5 Tastor_6
## Period1 A.2      C.2      C.1      B.1      B.2      A.1
## Period2 B.1      A.1      A.2      C.2      C.1      B.2
## Period3 C.2      B.2      B.1      A.1      A.2      C.1
## Period4 A.1      C.1      C.2      B.2      B.1      A.2
## Period5 B.2      A.2      A.1      C.1      C.2      B.1
## Period6 C.1      B.1      B.2      A.2      A.1      C.2
```

2.2.2 Analysis

The analysis on simulated data gives:

```
LS.rand$Y <- runif(36)
LS.aov <- aov(Y ~ tastor + period + recipe * temp, data = LS.rand)
summary(LS.aov)

##             Df Sum Sq Mean Sq F value Pr(>F)
## tastor       5 0.2146 0.04292   0.415  0.832
## period       5 0.2474 0.04948   0.479  0.788
## recipe       2 0.4049 0.20245   1.959  0.167
## temp         1 0.0451 0.04511   0.436  0.516
## recipe:temp  2 0.0957 0.04784   0.463  0.636
## Residuals   20 2.0671 0.10335
```

3 The main components of planor

3.1 Factors

In the usual R meaning, a factor is a column of a dataframe with a discrete set of levels. At the construction stage, the design dataframe has yet to constructed, so planor has the factor contains mainly the information on the set of levels.

There is a planor an S4 class to define a list of factors and to prepare it for design search.

3.2 Expected models and terms to estimate

Presently (17 January 2016, planor 0.2.4), the following rules are implemented in planor:

- all the factors declared in a model formula are treated as independent factors (as opposed to nested or correlated factors) when the formula is first processed. Besides, the model is supposed to be complete with respect to marginality (that is, it contains the general mean and, if it contains for example the interaction `B*C`, it must also contain the marginal terms `B` and `C`). It follows that no nesting relationship is allowed in the model formulae. Thus, `B+C`, `B*C`, `A:B` and `(A+B+C)^2` can be used (provided the whole model is complete), whereas `B %in% A`, `A/B` or the whole formula `~A+B+B:C` generate an error with a message about the problem.

```
test <- regular.design(factors = LETTERS[1:3], nlevels = c(2,
    3, 2), model = ~A + B/C, estimate = ~A, nunits = 12)

## Error in planor.modelterms(model): The factorial term B:C is present in the
model formula 1 but not all of its marginal terms.  This is not allowed for model
formulae.
```

- on the contrary, an estimate formula may be incomplete with respect to marginality. For example, it may contain the interaction `B*C` but not the marginal term `C`, which is interpreted as having interest in the interaction contrasts between B and C but not in the main effect of C. When this happens and because it is quite an unusual requirement, a warning is given.

```
test <- regular.design(factors = LETTERS[1:3], nlevels = c(2,
    3, 2), model = ~A * B * C, estimate = ~A + B/C, nunits = 12)

## Warning in planor.modelterms(model): The factorial term B:C is present in
the estimate formula 1 but not all of its marginal terms.  Consequently, this
or these marginal terms will be considered of no interest and may be unestimable
in the final design.

## The search is closed: max.sol = 1 solution(s) found
```

- if a model formula (respectively an estimate formula) is empty, then the associated estimate formula (respectively the associated model formula) are considered to provide the ineligible factorial terms (those that must not be confounded with the general mean). If both a model formula and its associated estimate formula are empty, the program stops with an error message.

```
test <- regular.design(factors = LETTERS[1:3], nlevels = c(2,
    3, 2), model = ~1, estimate = ~0, nunits = 12, base = ~A +
    B + C)

## Error in planor.ineligibleterms(modelterm.matrices): There is no factor at
all in the pair 1 of model and estimate formulae.  This is not allowed in PLANOR.
```

- in principle, the factorial terms in an estimate formula must be a subset of the factorial terms in the associated model formula. However, for some special uses not detailed here, it is not considered an error when this is not the case.

3.3 Experimental units

In planor, the main information to give about the experimental units is their number. Indeed, the `nunits` argument of `planor.designkey` or `regular.design` is compulsory in the present versions of the package.

The most recent versions give an explicit error message if `nunits` is missing or null, together with information on the number of units that would be required for a full factorial design. It is planned to give more help to the user on this point in the future versions of planor.

3.4 Key matrices

3.5 Design solutions

4 Construction of factorial designs: more examples

4.1 Full and fractional factorial designs

In a full factorial design, all possible combinations of the factors' levels are equireplicated. When there are many factors or many levels per factor, the full factorial design is usually not feasible because its size is larger than the maximum possible number of units. In that case, one can use a fractional design.

A fractional factorial design of odd resolution R guarantees that all effects of order strictly smaller than $R/2$ can be estimated, provided all factorial effects of order larger than 2 are assumed to be ngligible.

For simplicity, we consider in this subsection that there are no block factors, so that the designs must be completely randomized.

4.1.1 Full factorial design

We now consider a design for three factors A, B, C at 2, 2, 3 levels respectively. Suppose we want to estimate all the factorial effects. Then the model formula must include the main effects and all interactions of A, B, C. All possible combinations of A, B, C, must be present, so that the minimum size of the design is 12. Note that the other possible sizes are restricted to multiples of 12 in planor, because of orthogonality constraints.

Provided the experimenter wants a completely randomized design, the block structure is limited to the `UNITS` level. Then the R code is:

```
ABC.rand <- regular.design(factors = LETTERS[1:3], nlevels = c(2,
    2, 3), model = ~A * B * C, nunits = 12, randomize = ~UNITS,
    output = "data.frame")

## The search is closed: max.sol = 1 solution(s) found

print(ABC.rand)

##    A B C
## 1  2 1 1
## 2  2 2 2
## 3  1 2 3
## 4  1 2 1
## 5  2 2 1
## 6  1 1 3
## 7  1 1 2
## 8  2 1 2
## 9  1 2 2
## 10 2 2 3
## 11 1 1 1
## 12 2 1 3
```

Based again on simulated data, the analysis gives:

```
ABC.rand$Y <- runif(12)
ABC.aov <- aov(Y ~ A * B * C, data = ABC.rand)
summary(ABC.aov)

##             Df  Sum Sq Mean Sq
## A            1 0.11604 0.11604
## B            1 0.08376 0.08376
## C            2 0.00193 0.00096
## A:B          1 0.05517 0.05517
## A:C          2 0.16719 0.08359
## B:C          2 0.00571 0.00285
## A:B:C        2 0.11305 0.05652
```

4.1.2 Fractional factorial design of resolution 5

The package planor was originally conceived to search for highly fractionated factorial designs. For a given set of factors, fractional designs require much fewer units than the full factorial design but still allow to analyse the data with a given incomplete anova model, under the assumption that the factorial effects not in the model are negligible. They have been used for real experiments for a long time [3] and, more recently, for computer experiments (*e.g.* [5], [14]).

For example, a fraction of resolution 5 guarantees that all terms can be estimated from a model with all main effects and two-factor interactions. A fractional design of resolution 5 is generated below for 10 factors at 4 levels, assuming that $2^{10} = 1024$ units are available instead of $4^{10} = 1\ 048\ 576$ for a full factorial design:

```
FFD <- regular.design(factors = LETTERS[1:10], nlevels = 4,
    model = ~(A + B + C + D + E + F + G + H + I + J)^2,
    nunits = 2^10, output = "data.frame")

## The search is closed: max.sol = 1 solution(s) found

print(dim(FFD))

## [1] 1024   10

print(FFD[1:5, ])

##   A B C D E F G H I J
## 1 1 1 1 1 1 1 1 1 1 1
## 2 1 1 1 1 1 2 2 2 2 2
## 3 1 1 1 1 1 3 3 3 3 3
## 4 1 1 1 1 1 4 4 4 4 4
## 5 1 1 1 2 2 1 1 2 2 4
```

Note that in planor syntax, it is equivalent, but shorter, to specify `resolution = 5` rather than

`model = ~(A+B+C+D+E+F+G+H+I+J)^2.`

5 Construction of regular factorial designs through the search for a design key

We now adopt a more progressive way to construct the design. For this reason, we focus on the `planor.designkey` function rather than `regular.design`. In that case, design construction involves two main steps :

1. the search for key matrices (function `planor.designkey`);
2. then the design generation and randomization (function `planor.design`).

We start by a short technical subsection. It cannot go into details, but we hope it helps to make a link with other approaches to the construction of regular factorial designs.

5.1 A very short technical point

A key matrix of base p in planor is a matrix of integers modulo p, where p is a prime. It encodes the information required to construct a regular factorial design for factors at p levels.

Consider for example a design for 4 factors A, B, C, D at $p = 2$ levels in $2^3 = 8$ units, whereas a full factorial design would require $2^4 = 16$ units. It is possible to construct a design which allows to estimate the main effects of the factors assuming the three- and four-factor interactions are negligible. The solution is explained in many books on factorial designs (*e.g.* [3]) :

- assimilate the factors' levels to $0, 1$ mod 2;
- make a full factorial design on A, B, C;
- add the level of D on each unit by the equation $D = A + B + C$ mod 2, called the *defining relationship* of the design.

Then it can be shown that the interaction $A.B.C.D$ is confounded with the general mean, the main effect A is confounded with the interaction $B.C.D$, etc.

In planor, this construction is encoded in the following key matrix of base 2 :

$$K = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The rows of K are associated with three factors U_1, U_2, U_3 which are called the units factors. The idea is that the set of units can be identified to the full factorial design on these units factors. The columns of K are associated with the treatment factors A, B, C, D. Here the first column of K means that in the design, we must have $A = U_1$ (modulo 2). The second, third and fourth columns mean $B = U_2$, $C = U_3$ and $D = U_1 + U_2 + U_3$, respectively. It follows that the defining relationship $D = A + B + C$ mod 2 will be satisfied.

The core algorithm in planor basically constructs K by searching for its columns successively, using a backtrack algorithm. However, there is also much pre-processing to turn the factors and model specifications into appropriate constraints on the columns of K. In particular, all factors are automatically decomposed into pseudofactors which all have a prime number of levels, and the whole problem is decomposed according to the different prime numbers involved.

A detailed presentation of the methodology implemented in planor is under preparation. See also the references given in the introduction or [16] for the extension of regular factorial designs to the case when different primes are involved.

5.2 Fractional designs with 2-level factors

5.2.1 Search for a design key

Consider an experiment to study four treatment factors A, B, C, D at two levels, using two blocks of size four. A full factorial design on the treatment factors would require 16 units. Only eight are available so that a fractional design must be used. In addition, some treatment effects are necessarily confounded with the block effect.

At first, we may look for a design adapted to the model that includes the main effects of the block and treatment factors, as well as the interactions between pairs of treatment factors :

```
ex1Key <- planor.designkey(factors=c("block","A","B","C","D"),nlevels=rep(2,5),
                           model=~block+(A+B+C+D)^2,
                           nunits=2^3)

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 2 to 5
##       first visit to column 2
##       first visit to column 3
##     ---    col. 3 ( j = 2) 4 selected candidates
##       first visit to column 4
##     ---    col. 4 ( j = 3) 1 selected candidates
##       first visit to column 5
##     ---    col. 5 ( j = 4) 0 selected candidates
##     ---    col. 4 ( j = 3) 1 selected candidates
##     ---    col. 5 ( j = 4) 0 selected candidates
##     ---    col. 3 ( j = 2) 4 selected candidates
## The search is closed: 0 solutions found
```

It turns out that planor fails to find a solution. There is indeed no solution to the problem.

For the second try, we keep the same model but relax the implicit constraint to estimate all factorial terms in the model. This is done by using the `estimate` argument of the `planor.designkey` function. This argument is optional : by default, it is considered that all terms in the `model` formula must be estimated. In contrast, we only require below that the main effects of the treatment factors be estimable. It follows that we now allow for designs in which two-factor interactions are mutually confounded.

```
ex1Key <- planor.designkey(factors=c("block","A","B","C","D"),nlevels=rep(2,5),
                           model=~block+(A+B+C+D)^2,
                           estimate=~A+B+C+D,
                           nunits=2^3)

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 2 to 5
##       first visit to column 2
##       first visit to column 3
##     ---    col. 3 ( j = 2) 5 selected candidates
##       first visit to column 4
##     ---    col. 4 ( j = 3) 4 selected candidates
##       first visit to column 5
##     ---    col. 5 ( j = 4) 1 selected candidates
## The search is closed: max.sol = 1 solution(s) found
```

During the search, the backtrack algorithm looks successively for new columns to add to the key matrix. Succinct information is given to check the algorithm progress (default argument `verbose=TRUE`). The search stops as soon as all columns of the key matrix have been found (default argument `max.sol=1`).

An alternative to using `planor.designkey` directly is to provide the information on the experiment step by step with the functions `planor.factors` and `planor.model`. The idea is to store the results of these functions in R objects and use them as arguments to `planor.designkey`. This may be convenient, for example, when one wants to explore several possible models and design sizes with the same set of factors.

```
ex1Fac <- planor.factors(factors=c("block","A","B","C","D"), nlevels=rep(2,5),
                         block=~block)
ex1Mod <- planor.model( model=~block+(A+B+C+D)^2, estimate=~A+B+C+D )
ex1Key <- planor.designkey(factors=ex1Fac, model=ex1Mod, nunits=2^3)

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 2 to 5
##       first visit to column 2
##       first visit to column 3
##     ---    col. 3 ( j = 2) 5 selected candidates
##       first visit to column 4
##     ---    col. 4 ( j = 3) 4 selected candidates
##       first visit to column 5
##     ---    col. 5 ( j = 4) 1 selected candidates
## The search is closed: max.sol = 1 solution(s) found
```

5.2.2 Design-key properties

In both cases, the key matrix solution is stored in the object `ex1Key`. Its detailed properties can be obtained by two different functions. The `summary` function prints the key matrix and the defining relationships associated with this key matrix. More detailed information on the aliasing between factorial effects is given by the function `alias`.

Note that we have used the optional `block` argument of `planor.factors` (also available in `planor.designkey`). It specifies the factors that should be considered as block (or nuisance) factors. In planor, the distinction between *treatment* and *block* factors is taken into account when studying confounding and aliasing properties.

```
summary(ex1Key, show="dtb")

##
## ********** Prime  2  design **********
##
## --- Solution  1  for prime  2  ---
##
## DESIGN KEY MATRIX
##     block A  B  C  D
## *U*     1  0  1  0  1
## *U*     0  1  1  0  0
## *U*     0  0  0  1  1
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A B C D
##
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = block A B
## 1 = block C D
```

```
alias(ex1Key)
```

```
##
## ********** Prime  2  design **********
##
## --- Solution  1  for prime  2  ---
##
## UNALIASED TREATMENT EFFECTS
## A ; B ; C ; D
##
## ALIASED TREATMENT EFFECTS
## A:C = B:D
## A:D = B:C
##
## TREATMENT AND BLOCK EFFECTS CONFOUNDED WITH BLOCK EFFECTS
## block = A:B = C:D
##
## UNALIASED BLOCK EFFECTS
## nil
##
##
## --- Synthesis on the aliased treatment effects for prime  2  ---
##
##      unaliased trt.aliased blc.aliased
## [1,]         4           4           2
```

5.2.3 Design generation

Last but not least, a design can be generated by the function `planor.design`. The design itself is the object in slot `design` of the more complex object generated by `planor.design`. An option allows the design to be randomized, according to a block structure formula that the user must specify (option `randomize`).

```
ex1Des <- planor.design(ex1Key)
print(getDesign(ex1Des))
```

```
##   block A B C D
## 1     1 1 1 1 1
## 2     1 1 1 2 2
## 3     1 2 2 1 1
## 4     1 2 2 2 2
## 5     2 1 2 1 2
## 6     2 1 2 2 1
## 7     2 2 1 1 2
## 8     2 2 1 2 1
```

```
ex1Rand <- planor.design(ex1Key, randomize=~block/UNITS)
print(getDesign(ex1Rand))
```

```
##   block A B C D
## 1     1 1 1 2 2
## 2     1 2 2 2 2
## 3     1 1 1 1 1
## 4     1 2 2 1 1
## 5     2 2 1 2 1
## 6     2 1 2 1 2
## 7     2 2 1 1 2
## 8     2 1 2 2 1
```

5.3 Fractional designs with 3-level factors

We keep the same example but with 3-level factors and a few more options. The results are not shown for sake of brevity.

```
# ***************** EXAMPLE 2 *****************
# Four 3-level treatment factors and one 3-level block factor
# Model: block+(A+B+C+D)^2  -   Estimate: A+B+C+D
# N = 3^3 = 27 units
#
ex2Key <- planor.designkey(factors=c(LETTERS[1:4],"block"),
                          nlevels=rep(3,5),
                          block=~block,
                          model=~block+(A+B+C+D)^2,
                          estimate=~A+B+C+D,
                          nunits=3^3, base=~A+B+C, max.sol=2)
summary(ex2Key)
summary(ex2Key)
ex2Des <- planor.design(ex2Key[2])
```

Two optional arguments of `planor.designkey` have been used, first to specify that A, B and C should be used as basic factors, and second to ask for two solutions whereas the default is one. Both solutions are examined by `summary` and the second one, say, is chosen by the user to generate a factorial design. When *basic* factors are specified, they are identified to the units factors U_i [8]. As a consequence, all combinations of the basic factors are guaranteed to be included in the design. When relevant, using basic factors is recommended because it can speed up the search.

The following lines also work; they illustrate that the basic factors need not be part of the model but they must have been declared in `planor.factors`.

```
ex2Fac <- planor.factors(factors=c(LETTERS[1:4], "block", "BASE"),
                        nlevels=rep(3,6) )
ex2Mod <- planor.model(model=~block+(A+B+C+D)^2,
                      estimate=~A+B+C+D )
ex2Key <- planor.designkey(factors=ex2Fac,
                          model=ex2Mod,
                          nunits=3^3,
                          base=~A+B+BASE,
                          max.sol=2)
```

5.4 Asymmetric fractional factorial designs

A regular fractional factorial design is called mixed or asymmetric when the numbers of levels of the factors involve several different prime numbers. The asymmetric designs constructed in planor consist of the cross products of designs based on each prime. This does not allow for a great flexibility in terms of confounding, but it enlarges the scope of situations that can be addressed.

```
# Four treatment factors at 6, 6, 4, 2 levels and one 6-level block factor
# Model: block+(A+B+C+D)^2 ; Estimate: A+B+C+D\n")
# N = 144 = 2^4 x 3^2 experimental units
mixKey <- planor.designkey(factors=c( LETTERS[1:4], "block"),
                           nlevels=c(6,6,4,2,6),
                           block=~block,
                           model=~block+(A+B+C+D)^2,
                           estimate=~A+B+C+D,
                           nunits=144,
                       base=~A+B+D, max.sol=2)

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 4 to 6
##       first visit to column 4
##       first visit to column 5
##     ---    col. 5 ( j = 2) 8 selected candidates
##       first visit to column 6
##     ---    col. 6 ( j = 3) 9 selected candidates
## The search is closed: max.sol = 2 solution(s) found
## Main step for prime p = 3 : key-matrix search
##   => search for column 3 .
##       first visit to column 3
## The search is closed: max.sol = 2 solution(s) found

summary(mixKey)

##
## ********** Prime  2  design **********
##
## --- Solution  1  for prime  2  ---
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_1 B_1 D C_1
##
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_1 B_1 block_1
## 1 = D C_1 block_1
##
## WEIGHT PROFILES
## Treatment effects confounded with the mean: 4^1
## Treatment effects confounded with block effects: 2^2
## Treatment pseudo-effects confounded with the mean: 4^1
## Treatment pseudo-effects confounded with block effects: 2^2
##
## --- Solution  2  for prime  2  ---
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_1 B_1 D C_1
##
```

```
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_1 D block_1
## 1 = B_1 C_1 block_1
##
## WEIGHT PROFILES
## Treatment effects confounded with the mean: 4^1
## Treatment effects confounded with block effects: 2^2
## Treatment pseudo-effects confounded with the mean: 4^1
## Treatment pseudo-effects confounded with block effects: 2^2
##
##
## ********** Prime  3  design **********
##
## --- Solution  1  for prime  3  ---
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## nil
##
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_2^2  B_2^2  block_2
##
## WEIGHT PROFILES
## Treatment effects confounded with the mean: none
## Treatment effects confounded with block effects: 2^1
## Treatment pseudo-effects confounded with the mean: none
## Treatment pseudo-effects confounded with block effects: 2^1
##
## --- Solution  2  for prime  3  ---
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## nil
##
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = A_2 B_2^2  block_2
##
## WEIGHT PROFILES
## Treatment effects confounded with the mean: none
## Treatment effects confounded with block effects: 2^1
## Treatment pseudo-effects confounded with the mean: none
## Treatment pseudo-effects confounded with block effects: 2^1
```

```
mixPlan <- planor.design(key=mixKey, select=c(1,1), randomize=~block/UNITS)
print(getDesign(mixPlan)[1:25,])
```

```
##     A B D C block
## 1   1 5 2 2     1
## 6   5 1 2 2     1
## 8   4 2 2 2     1
## 10  4 2 1 4     1
## 15  3 6 1 4     1
## 17  2 4 1 4     1
## 19  2 4 2 2     1
## 24  4 2 2 1     1
## 26  3 6 2 1     1
## 28   3 6 2 2      1
## 33   3 6 1 3      1
## 35   5 1 2 1      1
## 109  4 2 1 3      1
## 114  6 3 2 2      1
## 116  6 3 1 4      1
## 118  2 4 2 1      1
## 123  6 3 1 3      1
## 125  1 5 2 1      1
## 127  1 5 1 4      1
## 132  5 1 1 3      1
## 134  2 4 1 3      1
## 136  6 3 2 1      1
## 141  1 5 1 3      1
## 143  5 1 1 4      1
## 2    6 1 2 1      2
```

The algorithm starts by decomposing the factors into pseudofactors that all have a prime number of levels. Then it performs a similar decomposition of the `model` and `estimate` terms. After these initial steps, separate key-matrix searches are performed, one for each prime involved in the problem. The prime decompositions are automatic and transparent to the user. The recomposition when generating a design is transparent too. In contrast, most information on the search process and on the fraction properties are given according to the prime decompositions.

5.5 Split-plot designs

In a split-plot experiment, there are two treatment factors `variety` and `fert`, say, at m and n levels respectively. The block structure consists of r blocks each containing m sub-blocks of size n and the factor `variety` is constrained to be constant within sub-blocks.

In an orthogonal split-plot design, each variety occupies one sub-block of each block, and each sub-block contains the n distinct levels of factor `fert`. In planor, this design can be constructed by defining the block structure as a cross between a `block` and a `subblock` factor. The `hierarchy` argument is used to specify that `variety` must be constant within the combinations of `block` and `subblock`. Two model-estimate pairs are given to the `listofmodels` argument. First, the main effect of `fert` and the interaction between `fert` and `variety` must be estimable when blocks and sub-blocks are included in the model. Second, the main effect of `variety` must be estimable between sub-blocks, that is, when blocks but not sub-blocks are included in the model. The command below calculates the design key of a split-plot design with $r = 2$, $n = 2$, $m = 2$.

```
splitKey <- planor.designkey(factors=list(block=1:2,
                               subblock=1:2,
                               variety=LETTERS[1:2],
                               fert=c("organic","mineral")),
                             block=~block+subblock,
                             hierarchy=list(~variety/(block*subblock)),
                             listofmodels=
                             list(c( ~block*subblock+variety*fert, ~fert+fert:variety),
                                  c( ~block+variety,               ~variety)),
                             nunits=2*2*2,
                             base=~block+subblock)

## Warning in planor.modelterms(model):  The factorial term fert:variety is present
in the estimate formula 1 but not all of its marginal terms.  Consequently, this or
```

```
these marginal terms will be considered of no interest and may be unestimable in the
final design.
```

```r
summary(splitKey)
```

```r
alias(splitKey)
```

```
print(getDesign(planor.design(splitKey, randomize=~block/subblock/UNITS)))

##   block subblock variety    fert
## 1     1        1       A mineral
## 2     1        1       A organic
## 3     1        2       B organic
## 4     1        2       B mineral
## 5     2        1       A mineral
## 6     2        1       A organic
## 7     2        2       B mineral
## 8     2        2       B organic
```

An alternative command to get the split-plot is given below. The main difference is that the `subblock` factor now takes *rm* levels and is considered as nested in `block` rather than crossed with it.

```
splitKey <- planor.designkey(factors=list(block=1:2,
                               subblock=1:4,
                               variety=LETTERS[1:2],
                               fert=c("organic","mineral")),
                             block=~block+subblock,
                             hierarchy=list( ~block/subblock, ~variety/subblock),
                             listofmodels=
                             list(c( ~subblock+variety*fert, ~fert+fert:variety),
                                  c( ~block+variety,              ~variety)),
                             nunits=2*2*2,
                             base=~subblock)

## Warning in planor.modelterms(model):  The factorial term fert:variety is present
in the estimate formula 1 but not all of its marginal terms.  Consequently, this or
these marginal terms will be considered of no interest and may be unestimable in the
final design.

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 3 to 5
##       first visit to column 3
##       first visit to column 4
##     ---    col. 4 ( j = 2) 2 selected candidates
##       first visit to column 5
##     ---    col. 5 ( j = 3) 4 selected candidates
## The search is closed: max.sol = 1 solution(s) found

print(getDesign(planor.design(splitKey, randomize=~block/subblock/UNITS)))

##   subblock block variety    fert
## 1        1     1       A organic
## 2        1     1       A mineral
## 3        2     1       B mineral
## 4        2     1       B organic
## 5        3     2       A organic
## 6        3     2       A mineral
## 7        4     2       B mineral
## 8        4     2       B organic
```

5.6 Fractional designs with nested factors and a complex block structure

We now consider an experiment with concrete and more complex specifications. This example stems from an experiment to study the cleaning of surfaces by a robot, see [8], example 3 on page 3. There are five treatment factors at 2 levels. The block structure consists of four plates with 2 rows and 4 columns per plate, resulting in 32 experimental units. In addition, the design must cope with experimental constraints between treatment and block factors. The treatment factors concentration (`conc`) and temperature (`Tact`) must remain constant within a plate. The treatment factors denoted by `nsoil` and `qsoil` must remain constant within each column of each plate. Only treatment factor rugosity (`Rug`) can be modified freely between experimental units.

To begin with, we show how to specify user-defined factor levels, by providing a list to the `factors` argument of `planor.factors`. Then, experimental constraints are specified through the `hierarchy` argument of `planor.factors`.

```
# ************ ROBOT1A EXAMPLE *************
# Block structure: 4 plates / (2 rows x 4 columns)
# Treatments: 4 2-level factors
# Hierarchy 1: conc constant in plate
# Hierarchy 2: Tact constant in plate
# Hierarchy 3: nsoil constant in plate x column
# Hierarchy 4: qsoil constant in plate x column
# N = 32 units
#
robotFac <- planor.factors( factors=list(
                             conc=c(1,3),
                             Tact=c(15,30),
                             nsoil=c("curd","Saint-Paulin"),
                             qsoil=c("0.01g","0.10g"),
                             Rug=c(0.25,0.73),
                             plate=1:4,
                             row=1:2,
                             col=1:4),
                            hierarchy=list(~conc/plate,
                            ~Tact/plate,
                            ~nsoil/(plate*col),
                            ~qsoil/(plate*col)))
```

This example requires several model-estimate combinations. The main model-estimate pair contains all the treatment factorial effects but no block effect. It guarantees that all treatment combinations will be present in the design, since all treatment factorial effects are required to be estimable in the model with no block effect. The second model-estimate pair (`listofmodels` argument) ensures that the `Rug` factor is orthogonal to block factors.

```
robotMod <- planor.model( model=~nsoil*qsoil*Rug*conc*Tact,
                          listofmodels=list(c(~plate+row+col+Rug, ~Rug)) )
```

The `base` option of the `planor.designkey` function is used here to impose that experimental units be associated with the combinations of the block factors.

```
robotKey <- planor.designkey(factors=robotFac, model=robotMod,
                             nunits=32, base=~plate+row+col)

## Preliminary step 1 : processing the model specifications
## Preliminary step 2 : performing prime decompositions on the factors
## Main step for prime p = 2 : key-matrix search
##   => search for columns 6 to 10
##       first visit to column 6
##       first visit to column 7
##     ---    col. 7 ( j = 2) 2 selected candidates
##       first visit to column 8
##     ---    col. 8 ( j = 3) 12 selected candidates
##       first visit to column 9
##     ---    col. 9 ( j = 4) 8 selected candidates
##       first visit to column 10
##     ---    col. 10 ( j = 5) 15 selected candidates
## The search is closed: max.sol = 1 solution(s) found
```

```
summary(robotKey[1])
```

```
##
## ********** Prime  2  design **********
##
## DESIGN KEY MATRIX
##         plate_1 plate_2 row col_1 col_2 conc Tact nsoil qsoil Rug
## plate_1       1       0   0     0     0    1    0     0     0   1
## plate_2       0       1   0     0     0    0    1     0     0   0
## row           0       0   1     0     0    0    0     0     0   1
## col_1         0       0   0     1     0    0    0     1     0   0
## col_2         0       0   0     0     1    0    0     0     1   0
##
## TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## 1 = plate_1 conc
## 1 = plate_2 Tact
## 1 = col_1 nsoil
## 1 = col_2 qsoil
## 1 = plate_1 plate_2 conc Tact
## 1 = col_1 col_2 nsoil qsoil
## 1 = plate_1 row Rug
## 1 = row conc Rug
## 1 = plate_1 col_1 conc nsoil
## 1 = plate_2 col_1 Tact nsoil
## 1 = plate_1 col_2 conc qsoil
## 1 = plate_2 col_2 Tact qsoil
## 1 = plate_1 plate_2 row Tact Rug
## 1 = plate_1 plate_2 col_1 conc Tact nsoil
## 1 = plate_1 plate_2 col_2 conc Tact qsoil
## 1 = plate_1 col_1 col_2 conc nsoil qsoil
## 1 = plate_2 col_1 col_2 Tact nsoil qsoil
## 1 = plate_2 row conc Tact Rug
## 1 = plate_1 row col_1 nsoil Rug
## 1 = row col_1 conc nsoil Rug
##
## The first 20 columns on a total of 31
##
## BLOCK-and-TREATMENT EFFECTS CONFOUNDED WITH THE MEAN
## nil
##
## WEIGHT PROFILES
## Treatment effects confounded with the mean: 2^4 3^4 4^5 5^9 6^5 7^3 8^1
## Treatment effects confounded with block effects: none
## Treatment pseudo-effects confounded with the mean: 2^4 4^6 3^2 5^6 6^4 8^1 7^6 9^2
## Treatment pseudo-effects confounded with block effects: none

robotDes <- planor.design(robotKey[1], randomize=~plate/(row*col))
print(getDesign(robotDes))

##    plate row col conc Tact        nsoil qsoil  Rug
## 1      1   1   1    3   15 Saint-Paulin 0.01g 0.73
## 2      1   1   2    3   15         curd 0.10g 0.73
## 3      1   1   3    3   15 Saint-Paulin 0.10g 0.73
## 4      1   1   4    3   15         curd 0.01g 0.73
## 5      1   2   1    3   15 Saint-Paulin 0.01g 0.25
## 6      1   2   2    3   15         curd 0.10g 0.25
## 7      1   2   3    3   15 Saint-Paulin 0.10g 0.25
## 8      1   2   4    3   15         curd 0.01g 0.25
## 9      2   1   1    1   15         curd 0.10g 0.25
## 10     2   1   2    1   15 Saint-Paulin 0.01g 0.25
## 11     2   1   3    1   15 Saint-Paulin 0.10g 0.25
## 12     2   1   4    1   15         curd 0.01g 0.25
## 13     2   2   1    1   15         curd 0.10g 0.73
## 14     2   2   2    1   15 Saint-Paulin 0.01g 0.73
## 15     2   2   3    1   15 Saint-Paulin 0.10g 0.73
## 16     2   2   4    1   15         curd 0.01g 0.73
## 17     3   1   1    1   30 Saint-Paulin 0.01g 0.73
## 18     3   1   2    1   30 Saint-Paulin 0.10g 0.73
## 19     3   1   3    1   30         curd 0.01g 0.73
## 20     3   1   4    1   30         curd 0.10g 0.73
## 21     3   2   1    1   30 Saint-Paulin 0.01g 0.25
## 22     3   2   2    1   30 Saint-Paulin 0.10g 0.25
## 23     3   2   3    1   30         curd 0.01g 0.25
## 24     3   2   4    1   30         curd 0.10g 0.25
## 25     4   1   1    3   30         curd 0.10g 0.73
## 26     4   1   2    3   30         curd 0.01g 0.73
## 27     4   1   3    3   30 Saint-Paulin 0.10g 0.73
## 28     4   1   4    3   30 Saint-Paulin 0.01g 0.73
## 29     4   2   1    3   30         curd 0.10g 0.25
## 30     4   2   2    3   30         curd 0.01g 0.25
## 31     4   2   3    3   30 Saint-Paulin 0.10g 0.25
## 32     4   2   4    3   30 Saint-Paulin 0.01g 0.25
```

Acknowledgements

The R package planor originates from the planor software which was written in the APL language by André Kobilinsky in 1995. This vignette was typed using the Sweave package (Leisch, 2002[13]).

References

[1] R. A. Bailey – "Factorial design and abelian groups", *Linear Algebra Appl.* **70** (1985), no. 349-368.

[2] — , *Design of comparative experiments*, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 2008.

[3] G. E. P. Box, W. Hunter & J. Hunter – *Statistics for experimenters*, Wiley, 1978.

[4] S. Cliquet, C. Durier & A. Kobilinsky – "Principle of a fractional factorial design for qualitative and quantitative factors: application to the production of *Bradyrhizobium japonicum* in culture media", *Agronomie* **14** (1994), p. 569–587.

[5] A. Courcoul, H. Monod, M. Nielen, D. Klinkenberg, L. Hogerwerf, F. Beaudeau & E. Vergu – "Modelling the effect of heterogeneity of shedding on the within herd *Coxiella burnetii* spread and identification of key parameters by sensitivity analysis", *Journal of Theoretical Biology* **284** (2011), p. 130–141.

[6] M. F. Franklin – "Selecting defining contrasts and confounded effects in p^{n-m} factorial experiments", *Technometrics* **27** (1985), p. 165–172.

[7] A. Kobilinsky – "Les plans factoriels", in *Plans d'expériences: applications à l'entreprise* (J. J. Droesbeke, J. Fine & G. Saporta, éds.), Technip, Paris, 1997, p. 69–209 (Chapter 3).

[8] A. Kobilinsky – "PLANOR : program for the automatic generation of regular experimental designs. version 2.2 for Windows", Tech. report, MIA Unit, INRA Jouy en Josas, 2005.

[9] A. Kobilinsky, A. Bouvier & H. Monod – "PLANOR : an R package for the automatic generation of regular fractional factorial designs. Version 1.0", Tech. report, MIA Unit, INRA Jouy en Josas, 2011.

[10] A. Kobilinsky & H. Monod – "Experimental design generated by group morphisms: an introduction", *Scand. J. Statist.* **18** (1991), p. 119–134.

[11] — , "Juxtaposition of regular factorial designs and the complex linear model", *Scand. J. Statist* **22** (1995), p. 223–254.

[12] A. Kobilinsky, H. Monod & R. A. Bailey – "Automatic generation of generalised regular factorial designs", Tech. report, International Newton Institute Cambridge, 2015.

[13] F. Leisch – "Sweave: Dynamic generation of statistical reports using literate data analysis", in *Compstat 2002 — Proceedings in Computational Statistics* (W. Härdle & B. Rönz, éds.), Physica Verlag, Heidelberg, 2002, ISBN 3-7908-1517-9, p. 575–580.

[14] A. Lurette, S. Touzeau, M. Lamboni & H. Monod – "Sensitivity analysis to identify key parameters influencing *Salmonella* infection dynamics in a pig batch", *Journal of Theoretical Biology* **258** (2009), no. 1, p. 43–52.

[15] H. D. Patterson & R. A. Bailey – "Design keys for factorial experiments", *Appl. Statist.* **27** (1978), p. 335–343.

[16] G. Pistone & M.-P. Rogantin – "Indicator function and complex coding for mixed fractional factorial designs", *Journal of Statistical Planning and Inference* **138** (2008), no. 3, p. 787 – 802.